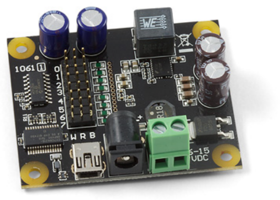
1061 User Guide



Go to this device's product page [1]

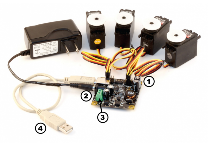
Getting Started

Checking the Contents

You should have received:	In order to test your new Phidget you will also need:
• A PhidgetAdvancedServo 8-Motor	• Some RC Servo Motors
• A Mini-USB Cable	• A 6 - 15V DC Power Supply (if it's a barrel connector it should have **center positive** polarity)

Connecting the Pieces

1. Connect the RC Servo Motors to the PhidgetAdvancedServo.
2. Plug in a power supply using the barrel connector.
3. You can also connect a power supply to the Terminal Block for high-current applications. Be sure to observe correct polarity.
4. Connect the PhidgetAdvancedServo to your computer using the USB cable.



Testing Using Windows 2000 / XP / Vista / 7

Make sure you have the current version of the Phidget library installed on your PC. If you don't, follow these steps:

1. Go to the Quick Downloads section on the Windows page
2. Download and run the Phidget21 Installer (32-bit, or 64-bit, depending on your system)
3. You should see the Ph icon on the right hand corner of the Task Bar.

Running Phidgets Sample Program

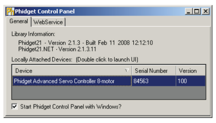
Double clicking on the Ph icon loads the Phidget Control Panel; we will use this program to ensure that your new Phidget works properly.

The source code for the **AdvancedServo-full** sample program can be found in the quick downloads section on the C# Language Page. If you'd like to see examples in other languages, you can visit our Languages page.

Updating Device Firmware

If an entry in this list is red, it means the firmware for that device is out of date. Double click on the entry to be given the option of updating the firmware. If you choose not to update the firmware, you can still run the example for that device after refusing.

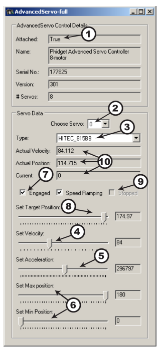
Double Click on the Ph icon to activate the Phidget Control Panel and make sure that the **Phidget Advanced Servo Controller 8-Motor** is properly attached to your PC.


Phidget Control Panel
General | WebService
Library Information:
Phidget21 - Version 2.1.3 - Built Feb 11 2008 12:12:10
Phidget21.NET - Version 2.1.3.11
Locally Attached Devices: (Double click to launch UI)

Device	Serial Number	Version
Phidget Advanced Servo Controller 8-motor	84563	100

☑ Start Phidget Control Panel with Windows?

1. Double click on Phidget Advanced Servo Controller 8-Motor in the Phidget Control Panel to bring up AdvancedServo-full and check that the box labelled Attached contains the word True.
2. Select a connected servo. In this example, a servo is connected at position 0.
3. Select your servo type. If your servo is not in the list, select “default”.
4. Use the Velocity slider to set the velocity limit. The servo will try to accelerate to this point during motion.
5. Use the Acceleration slider to set the acceleration.
6. Use the Min/Max Position slider to set the position range. It can prevent the servo from trying to go beyond its actual range of motion.
7. Check the Engaged box to power the servo. If the servo is not already the target position, it should begin to move.
8. Move the Position slider to set a target position. The servo will turn until its actual position equals the target position. If Speed Ramping is enabled, the servo will move using the user set acceleration and velocity.
9. When the servo has reached the target position, a tick mark will appear in the Stopped box.
10. These boxes report the controller’s internally calculated position and velocity of the servo, as well as current consumed in amps.


AdvancedServo-full
AdvancedServo Control Details
Attached: True
Name: Phidget Advanced Servo Controller 8-motor
Serial No.: 177825
Version: 301
Servos: 8
Servo Data
Choose Servo: 0
Type: HITEC_815BB
Actual Velocity: 84.112
Actual Position: 114.715
Current: 0
☑ Engaged ☑ Speed Ramping ☐ Stopped
Set Target Position: 174.97
Set Velocity: 84
Set Acceleration: 296797
Set Max position: 180
Set Min Position: 0

Testing Using Mac OS X

1. Go to the Quick Downloads section on the Mac OS X page
2. Download and run the Phidget OS X Installer
3. Click on System Preferences >> Phidgets (under Other) to activate the Preference Pane
4. Make sure that the Phidget Advanced Servo Controller 8-Motor is properly attached.
5. Double Click on Phidget Advanced Servo Controller 8-Motor in the Phidget Preference Pane to bring up the AdvancedServo-full Sample program. This program will function in a similar way as the Windows version.

Using Linux

For a step-by-step guide on getting Phidgets running on Linux, check the Linux page.

Using Windows Mobile / CE 5.0 / CE 6.0

For a step-by-step guide on getting Phidgets running on Windows CE, check the Windows CE page.

Technical Details

An RC Servo can be instructed to move to a desired position by the controller. Internally, it monitors the current position, and drives the motor as fast as it can until it reaches the desired position. This is a very cheap and simple way to control a motor. It has some limitations - there is no way for the controller to know the current position and speed of the motor. Applications that want smooth movement suffer from the aggressive acceleration.

The PhidgetAdvancedServo is able to address some of these limitations. Instead of sending the desired position immediately, the PhidgetAdvancedServo sends a series of progressive positions according to acceleration and velocity parameters. In most applications, this dramatically smooths the operation of the servo, and allows reasonably precise control of position, velocity and acceleration. The PhidgetAdvancedServo has a built in switching regulator - this allows it to efficiently operate from a wide voltage range (6-15VDC), and maintain proper power to the servo motors even if the power supply is varying. This built in voltage regulator will not operate if your power supply is undersized.

For more information about servo motors and controllers, check out the Servo Motor and Controller Primer.

Current Sense

The PhidgetAdvancedServo continuously measures the current consumed by each motor. The current roughly corresponds to torque, making it possible to detect several scenarios.

- By monitoring for no current, it's possible to determine if the servo is not connected. It may not be possible to distinguish between a servo at rest and a servo not attached.
- Stalled motors can be detected, by monitoring for the maximum current possible with your motor.
- The position limits of the servo can be programmatically determined by moving the servo until it stalls against the internal or external stops.

The 1061 specifications state that the board is suited to a maximum 1.6A continuous draw per channel to a maximum of 12A for the whole board. There is not overcurrent shutoff on a per channel basis though, only for the whole board. This means that technically you could put 12A through one channel if you had a big enough servo or were short circuiting one of the channels. This would burn out the current sensing electronics very quickly. As a result, monitoring the current levels in your code is a good idea as you can implement a safety shutoff yourself.

Limitations

The PhidgetAdvancedServo does not know the current position of the motor on its own. If your motor is free to move, and is not being driven beyond the physical limitations of the motor, the position returned to your application will be very close to the position of the motor.

Degree Abstraction

The PhidgetAdvancedServo software component uses degrees to specify position, velocity, and acceleration. The degree unit is translated into a pulse sent to the servo, but it's up to the servo to translate this signal into a particular position. This translation varies between servo models and manufacturers, and it is up to the user to set up their particular servo so that the degree abstraction matches up with reality. Phidgets Inc. has quantified a number of common servo motors (see API section), which can be used with the ServoType function for to set these parameters. For servos not in this list, or to quantify a specific servo, the setServoParameters function can be used.

Defining a Custom Servo

Servos are driven with a PCM (Pulse Code Modulation) signal. To define a custom servo, you need to find the minimum and maximum PCM values that the servo supports.

The easiest way to do this is by bringing up the example and choosing RAW_us_MODE. This will display all positions in microseconds instead of degrees. Move the servo to both of its extremes, stopping when it hits the stops, then easing up a little (leave a few degrees of leaway), and record these values. You could also choose a specific range in degrees that you require and find the PCM values that correspond. Most servos operate within 500us - 2500us.

Record the degrees of rotation that this PCM range represents (using a protractor, for example).

Calculate the maximum velocity of your servo, in degrees/second. Most servos list their max speed in seconds per 60 degrees. convert this into degrees/second:

$$\text{Velocity}(^o/s) = \frac{60}{\text{Velocity}(\frac{\text{s}}{60^o})}$$

The actual maximum velocity of your servo may be slightly higher or lower, as velocity depends on voltage.

Feed these four values into the serServoParameters function to complete the set up. This should be done before any other function are called (in the attach event ideally).

Note that many servos can operate quite a bit outside of their rated ranges.

Degree Abstraction (Historical Model)

Historically, our degree abstraction has been based on the Futaba FP-S148 servo. This is the default abstraction used for the PhidgetAdvancedServo, to maintain backwards compatibility when the new model was added.

$$\text{PWM(ms)} = (\text{degrees} + 23) * \frac{4}{375}$$

Using the 1061 with a Servo Motor

The PhidgetAdvancedServo has been designed to be used with a variety of RC servo motors independent of the motor-specific position, velocity and torque limits. Select a motor that suits your application and falls within the PhidgetAdvancedServo device specifications.

To use a servo motor, first select (in software) which attached motor the PhidgetAdvancedServo should affect. Position, velocity and acceleration can be controlled for each individual motor. The software can also display a readout of the electrical current flowing through each motor.

Using the 1061 with Continuous Rotation Servos

A continuous rotation servo is a servo motor that has had its headgear-stop removed and potentiometer replaced by two matched-value resistors. This has the effect of allowing the motor to rotate freely through a full range of motion, but disables the motor’s ability to control it’s position.

When using the PhidgetAdvancedServo with a servo motor modified in this way, the position control in software becomes the motor’s speed control. Because the two resistors that replace the motor’s potentiometer are matched in value, the motor will always think its shaft is at center position. If the target position in software is set to center, the motor will believe it has achieved the target and will therefore not rotate. The further away from center the target position is set to, the faster the motor will rotate (trying to reach that position, but never doing so). Changing the value above or below center changes the direction of rotation.

Using the 1061 with Electronic Speed Controllers (ESCs)

Electronic Speed Controllers are commonly used in RC hobby planes, cars, helicopters. It’s a controller that accepts a PWM input signal, and controls a motor based on that signal. The ESC accepts power from an external source, normally a battery pack.

ESCs can be controlled by the 1061, but the vast majority of ESCs on the market will destroy the 1061 if they are plugged in without modification. In a hobby RC system, the ESC is responsible for regulating some of the battery current down to ~5V, and supplying it to the radio receiver. An ESC designed to the power the receiver will advertise that it has a Battery Eliminator Circuit (BEC). When you plug an ESC into the 1061, the 1061 is acting as the radio receiver. The 1061 was not designed to be powered by the devices it controls, and the voltage regulator on the 1061 will self-destruct if a device tries to power it. If the center pin from the 3-wire servo connector between the 1061 and the ESC is disconnected, the BEC on the ESC will not be able to power the 1061, and the voltage regulator will not fail.

How the ESC inteprets the PWM signal and controls the motor is a function of the ESC. Higher end ESCs can be configured based on the application.

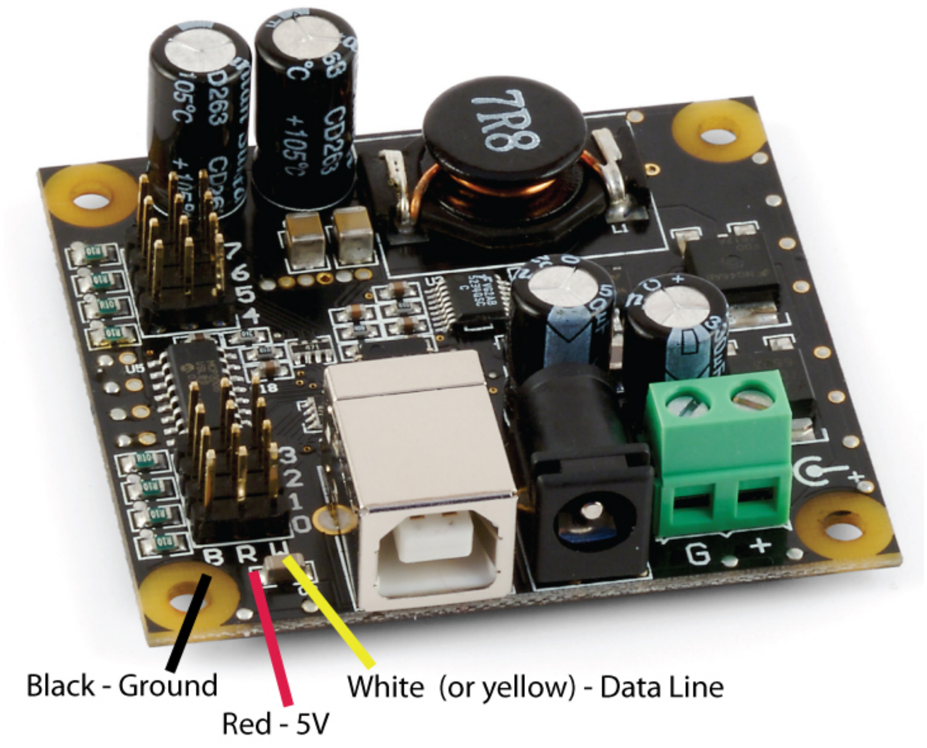
The hobby RC market has transitioned to Brushless DC Motors (BLDC). As you select an ESC, watch that the battery voltage input matches that of your system, and the type of motor controlled is what you have. Brushed DC and Brushless DC Motors are completely different, and require different controllers.

Wiring layout is critical with ESCs. The currents to the motor and on the ground return can be enormous. If these currents end up travelling back through USB cables, the system will not be stable. Some ESCs are optically isolated (OPTO) - a big advantage that reduces interference.

Synchronization of Multiple Servo Motors

Many applications call for several servo motors operating in unison - for example, operating a CNC table, or a robot arm. Highly precise synchronization of multiple servos using the 1061 is not possible, as the sequencing will be affected by the real-time performance of your operating system. Each servo is controlled as a independent unit, so there is no way of arranging for a particular action to happen to all motors at the same time. Typical jitter can be 10-30mS.

Connecting your Servo Motor to the 1061



The pins on the 1061 are labelled B R W on the board:

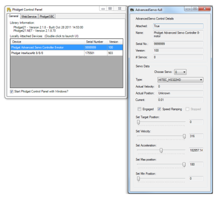
- B for Black is the Ground
- R for Red is 5V
- W for White (or Yellow depending on your servo motor) is the Data Line

If you need to use a servo motor with a higher power requirement than the 1061 can provide, you can cut the red and black wires on the sensor cable and solder them to the power and ground of an external power supply. If you do this, make sure you disconnect the red wire from the 1061, because reverse current from the external power supply could damage the servo controller board.

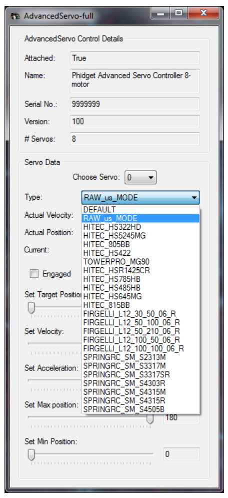
Servo Calibration

Each servo behaves slightly differently. The Phidgets API provides calibration parameters for a number of different models of servo motors; however these values are generic with respect to the model and not specific to each individual unit. It is possible to generate your own values through some manual calibration in the event that your application requires very accurate control or if you are using a servo for which default values are not provided. This is done as follows:

Open the Phidget control panel and open the GUI for the servo controller.



Plug your servo in and set the servo type to RAW_us_MODE.



AdvancedServo-full

AdvancedServo Control Details

Attached: True

Name: Phidget Advanced Servo Controller 8-motor

Serial No.: 9999999

Version: 100

Servos: 8

Servo Data

Choose Servo: 0

Type: RAW_us_MODE

DEFAULT
RAW_us_MODE
HITEC_HS322HD
HITEC_HS5245MG
HITEC_805BB
HITEC_HS422
TOWERPRO_MG90
HITEC_HSR1425CR
HITEC_HS785HB
HITEC_HS485HB
HITEC_HS645MG
HITEC_815BB
FIRGELLI_L12_30_50_06_R
FIRGELLI_L12_50_100_06_R
FIRGELLI_L12_50_210_06_R
FIRGELLI_L12_100_50_06_R
FIRGELLI_L12_100_100_06_R
SPRINGRC_SM_S2313M
SPRINGRC_SM_S3317M
SPRINGRC_SM_S3317SR
SPRINGRC_SM_S4303R
SPRINGRC_SM_S4315M
SPRINGRC_SM_S4315R
SPRINGRC_SM_S4505B

Actual Velocity:

Actual Position:

Current:

Engaged

Set Target Position:

Set Velocity:

Set Acceleration:

Set Max position: 180

Set Min Position: 0

Choose the correct servo motor (based on what pins the motor is connected to) and then starting with the target position slider in the approximate middle, engage the motor. The motor should move into position.

Move the target position slider to the left until the motor stops turning. The servo should stop before the slider hits its limit. Take note of the value at which the motor stops, the more precise you can be the better but the nearest multiple of 10 is usually sufficient. This is the low µs value and will be used later.

Move the slider to the right and again take note of where the servo stops turning. This will be the high µs value.

Using a protractor, measure the actual range of movement of the servo motor. This should be close to the rated value but it will likely not match exactly. Accuracy to the nearest degree is good enough.

To do this start with the motor at one extreme. Mark a spot on the motors output shaft in line with 0 degrees and then rotate the motor to its other extreme. Take the value of where the mark you made now rests and that is the motor's full range of motion. For linear actuators, retract the arm all the way, mark the position of the end of the arm, extend it to its maximum length, and then measure the distance in millimeters from the end of the arm to the old position.

Finally, check the motor's data sheet to see the maximum rated velocity. Now that all the measurements have been taken, you can use

```
setServoParamters(min_us, max_us, degrees, velocity_max)
```

setSevoParameters has 4 arguments, min_us is the minimum µs value you recorded, max_us is the max µs value, degrees is the range measured with the protractor, and velocity_max is the maximum rated velocity from the data sheet in degrees per second.

For example:

The HITEC HS-322HD used in testing has the following parameters:

HITEC HS-322HD Parameters

Argument	Value
min_us	630
max_us	2310
degrees	180
velocity_max	316

So for example, in C# to initialize the specific HS-322HD tested the following change would be made to the AdvancedServo example code in the advServo_Attach function:

```
        void advServo_Attach(object sender, AttachEventArgs e)
        {
            ...
            //Set the default servo type to the one Phidgets sells
            foreach (AdvancedServoServo motor in attached.servos)
                //motor.Type = ServoServo.ServoType.HITEC_HS322HD;
                  motor.setServoParameters(630, 2310, 9, 2949);
            ...
        }
```

Your program has now been calibrated to your servo motor. You can get calibration parameters for multiple motors and use all of them in one program should you require it.

API

We document API Calls specific to this product in this section. Functions common to all Phidgets and functions not applicable to this device are not covered here. This section is deliberately generic. For calling conventions under a specific language, refer to the associated API manual in the Quick Downloads section for that language. For exact values, refer to the device specifications.

Data Structures

enum Phidget_ServoType {

PHIDGET_SERVO_DEFAULT = 1,

PHIDGET_SERVO_RAW_us_MODE,

PHIDGET_SERVO_HITEC_HS322HD,

PHIDGET_SERVO_HITEC_HS5245MG,

PHIDGET_SERVO_HITEC_805BB,

PHIDGET_SERVO_HITEC_HS422,

PHIDGET_SERVO_TOWERPRO_MG90,

PHIDGET_SERVO_HITEC_HS1425CR,

PHIDGET_SERVO_HITEC_HS785HB,

PHIDGET_SERVO_HITEC_HS485HB,

PHIDGET_SERVO_HITEC_HS645MG,

PHIDGET_SERVO_HITEC_HS815BB,

PHIDGET_SERVO_USER_DEFINED

}

Used with the ServoType [get,set] functions. These are servos that have been quantified by Phidget Inc. for your convenience. The Default setting is included for historical reasons, so that the API will be backwards compatible by default. RAW_us_MODE is used for quantifying new servos, or simply when a microsecond based interface makes more sense then a degree based abstraction. USER_DEFINED should never be set directly with ServoType - this is returned when a custom servo type has been defined with setServoParameters.

Functions

int Count() [get]

Returns the number of servos this PhidgetAdvancedServo can control. In the case of the 1061, this will always return 8. This call does not return the number of servos actually connected.

double Acceleration(int ServoIndex) [get,set]

Acceleration is the maximum change in velocity the PhidgetAdvancedServo uses when speeding up / slowing down a servo.

- The range of valid Acceleration is bounded by AccelerationMax/AccelerationMin.
- There is a practical limit on how fast your servo can accelerate, based on load and the physical design of the motor.
- This property should always be set by the user as part of initialization. The value does not initialize to the value last set on the device.

double AccelerationMax(int ServoIndex) [get] : Constant

AccelerationMax is the upper limit to which Acceleration can be set. For the 1061, this will always return 320000.

double AccelerationMin(int ServoIndex) [get] : Constant

AccelerationMin is the lower limit to which Acceleration can be set. For the 1061, this will always return 19.53125.

double Velocity(int ServoIndex) [get]

Velocity returns the actual velocity that a particular servo is being driven at. A negative value means it is moving towards a lower position. This call does not return the actual physical velocity of the connected motor.

double VelocityLimit(int ServoIndex) [get, set]

Gets or sets the maximum absolute velocity that the PhidgetAdvancedServo controller will drive the servo. If it's changed mid-movement, the controller will accelerate accordingly. If the target position of the controller is near enough, then the VelocityLimit may never be reached.

- This property should always be set by the user as part of initialization.
- There is a practical limit on how fast your servo can rotate, based on the physical design of the motor.
- The range of VelocityLimit is bounded by VelocityMax/VelocityMin
- Note that when VelocityLimit is set to 0, the servo will not move.

double VelocityMax(int ServoIndex) [get] : Constant

VelocityMax is the absolute upper limit to which Velocity can be set. For the1061, this will always return 6400.

double VelocityMin(int ServoIndex) [get] : Constant

VelocityMin is the absolute lower limit to which Velocity can be set. For the 1061, this will always return 0.

double Position(int ServoIndex) [get,set]

Position is used for both the target and actual position for a particular servo. If the servo is currently engaged and a new value is set, then the controller will continuously try to move to this position. Otherwise, this call will return the current position of the servo. This call does not return the actual physical position of the servo.

- The range of Position is bounded by PositionMin/PositionMax
- If the servo is not engaged, then the position cannot be read.
- The position can still be set while the servo is not engaged. Once engaged, the servo will snap to position if it is not there already.
- This property should be set by the user as part of initialization. If not, it will report the last value set on the device (unless the 1061 has been power-cycled).
- Get will return the last value as reported by the device. This means sets to this value will take a small amount of time to propagate.

double PositionMax(int ServoIndex) [get,set]

PositionMax is the upper limit to which Position can be set, and is initialized to 233. It can be used to prevent the controller from going beyond a servo's range of motion. A PhidgetException will be thrown if this is set above 233 or below PositionMin.

double PositionMin(int ServoIndex) [get,set]

PositionMin is the lower limit to which Position can be set, and is initialized to -22.9921875. It can be used to prevent the controller from going beyond a servo's range of motion. A PhidgetException will be thrown if this is set below -22.9921875 or above PositionMax.

double Current(int ServoIndex) [get]

Current returns the power consumption in amps for a particular servo. The value returned for a disconnected or idle servo will be slightly above zero due to noise.

bool SpeedRamping(int ServoIndex) [get,set]

SpeedRamping enables or disables whether the PhidgetAdvancedServo tries to smoothly control the motion of a particular servo. If enabled, then the 1061 will progressively send commands based on velocity, acceleration and position.

- This property should be set by the user as part of initialization. If not, it will report the last value set on the device (unless the 1061 has been power-cycled).
- Get will return the last value as reported by the device. This means sets to this value will take a small amount of time to propagate.

bool Engaged(int ServoIndex) [get,set]

Enables a particular servo to be positioned. If this property is false, no power is applied to the motors. Note that when it is first enabled, the servo will snap to position, if it is not physically positioned at the same point.Engaged is useful for relaxing a servo once it's reached a given position. If you are concerned about keeping accurate track of position, Engaged should not be disabled until Stopped = True.

- This property should be set by the user as part of initialization. If not, it will report the last value set on the device (unless the 1061 has been power-cycled).
- Get will return the last value as reported by the device. This means sets to this value will take a small amount of time to propagate.

bool Stopped(int ServoIndex) [get]

Stopped returns false if the servo is currently in motion. It guarantees that the servo is not moving (unless you are moving it by hand), and that there are no commands in the pipeline to the servo. Note that virtually any API calls will cause Stopped to be temporarily false, even changing Acceleration or VelocityLimit on a stopped servo.

Phidget_ServoType ServoType(int ServoIndex) [get,set]

Gets / Sets the servo type for an index. There is a list of some common servos that have been predefined by Phidgets Inc. This sets the PCM range (range of motion), the PCM to degrees ratios used internally and the maximum velocity. This allows the degree based functions to be accurate for a specific type of servo.

Note that servos are generally not very precise, so two servos of the same type may not behave exactly the same. Specific servo motors, as well as servos not in the list, can be independently quantified by the user and set up with the setServoParameters funtion. This is detailed in the technical section.

void setServoParameters(int ServoIndex, double MinUs, double MaxUs, double Degrees, double VelocityMax)

Sets the parameters for a custom servo motor. MinUs is the minimum PCM in microseconds, MaxUs is the maximum PCM in microseconds, Degrees is the degrees of rotation represented by the given PCM range and VelocityMax is the maximum velocity that the servo can maintain, in degrees/second.

Quantifying a custom servo motor is detailed in the technical section.

Events

VelocityChange(int ServoIndex, double Velocity) [event]

An event issued when the velocity changes on a motor.

PositionChange(int ServoIndex, double Position) [event]

An event issued when the position changes on a motor.

CurrentChange(int ServoIndex, double Current) [event]

An event issued when the current consumed changes on a servo.

Product History

Date	Board Revision	Device Version	Comment
July 2008	0 [2]	300	Product Release
May 2011	0 [2]	301	getLabelString fixed for labels > 7 characters
January 2012	1 [3]	301	Mini-USB connector, larger input terminal blocks (12-24AWG)
February 2012	1 [3]	302	Prevent signal line startup pulses
February 2013	1 [3]	303	USB stack changes; was unstable with lots of set reports
August 2013	1 [3]	304	USB bug fix. Some OUT messages not getting through

References

[1] http://www.phidgets.com/products.php?product_id=1061
[2] http://www.phidgets.com/products.php?product_id=1061_0
[3] http://www.phidgets.com/products.php?product_id=1061_1

Article Sources and Contributors

1061 User Guide *Source*: http://www.phidgets.com/wiki/index.php?title=1061_User_Guide *Contributors*: Burley, Mparadis

Image Sources, Licenses and Contributors

Image:1061.jpg *Source*: http://www.phidgets.com/wiki/index.php?title=File:1061.jpg *License*: unknown *Contributors*: Burley

File:1061_0_Connecting_The_Hardware.jpg *Source*: http://www.phidgets.com/wiki/index.php?title=File:1061_0_Connecting_The_Hardware.jpg *License*: unknown *Contributors*: Mparadis

File:Ph.jpg *Source*: http://www.phidgets.com/wiki/index.php?title=File:Ph.jpg *License*: unknown *Contributors*: Mparadis

File:1061_0_Control_Panel_Screen.jpg *Source*: http://www.phidgets.com/wiki/index.php?title=File:1061_0_Control_Panel_Screen.jpg *License*: unknown *Contributors*: Mparadis

File:1061_0_Motor_Control_Screen.jpg *Source*: http://www.phidgets.com/wiki/index.php?title=File:1061_0_Motor_Control_Screen.jpg *License*: unknown *Contributors*: Mparadis

File:1061_0_Pins.jpg *Source*: http://www.phidgets.com/wiki/index.php?title=File:1061_0_Pins.jpg *License*: unknown *Contributors*: Mparadis

File:servocalibrate1.png *Source*: http://www.phidgets.com/wiki/index.php?title=File:Servocalibrate1.png *License*: unknown *Contributors*: Mparadis

File:servocalibrate2.png *Source*: http://www.phidgets.com/wiki/index.php?title=File:Servocalibrate2.png *License*: unknown *Contributors*: Mparadis